Olli Paakkunainen

Automation of Energy Usage with Network Devices

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Tehnology

Thesis

29 November 2014

Helsinki Metropolia University of Applied Sciences

Author	Olli Paakkunainen
Title	Automation of Energy Usage with Network Devices
Number of Pages	50 pages + 2 appendixes
Date	29 November 2014
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor	Erik Pätynen, Senior Lecturer

The goal of this thesis was to implement an easily deployable solution to reduce energy consumption in a corporate network. The solution was a simple script which can be installed to a network switch to automatically shut down connected devices.

First the technical background to the subject was examined and two network device manufacturers were chosen for testing. These manufacturers were Juniper Networks and Cisco Systems. A manufacturer provided script for Juniper switches was tested and a similar script for Cisco switches was implemented and tested. After testing the script deployment and functionality of the scripts, power calculations were done to measure resulted energy savings.

The outcome of this work was two working and tested scripts which reduce energy consumption in a corporate network. The thesis provides instructions to deploy the scripts and sample energy calculations to review the impact of these scripts if they are deployed in a company network.

The solution presented in this thesis can work as a part of a company's energy reduction plan. Scripts can be enabled with very little work in many different networks. Energy automation can then be extended to other devices and the scripts introduced in this thesis can be developed further or tailored to a company's special needs.

Keywords	energy automation, green ICT, data networks

Tekijä Otsikko Sivumäärä Aika	Olli Paakkunainen Energiansäästö verkkolaitteilla 50 sivua + 2 liitettä 29.11.2014
Tutkinto	insinööri (AMK)
Koulutusohjelma	tietotekniikka
Suuntautumisvaihtoehto	ohjelmistotekniikka
Ohjaaja	lehtori Erik Pätynen

Insinöörityön tavoitteena oli luoda helposti käyttöönotettava ratkaisu energinsäästöön yritysverkoissa. Ratkaisu oli yksinkertainen verkkokytkimiin asennettava komentosarja, joka automaattisesti sammuttaa kytkimiin kiinnitettyjä laitteita.

Ensin käsiteltiin aiheen tekninen tausta ja valittiin kaksi verkkolaitevalmistajaa testausta varten. Nämä verkkolaitevalmistajat olivat Juniper Networks ja Cisco Systems. Juniper kytkimillä testattiin valmistajan tarjoama komentosarja, jonka jälkeen samankaltainen komentosarja toteutettiin ja testattiin Ciscon kytkimellä. Komentosarjojen käyttöönoton ja toiminnallisuuden testaamisen päätyttyä, tehomittauksilla todettin mahdollinen energian säästö.

Tulokseksi saatiin kaksi toimivaa ja testattua komentosarjaa energiansäästöön yritysverkoissa. Insinöörityö tarjoaa ohjeet komentosarjojen käyttöönottoon, ja enegiansäästölaskelmia komentosarjojen vaikutusten tarkasteluun, jos ne otetaan käyttöön yritysverkossa.

Insinöörityön ratkaisu voi toimia osana yrityksen energiansäästösuunnitelmaa. Komentosarjat voidaan ottaa käyttöön hyvin pienellä työllä monissa erilaisissa verkoissa. Energiansäästöä voi tämän jälkeen laajentaa eri laitteisiin, ja insinöörityössä esiteltyjä komentosarjoja voidaan kehittää edelleen tai muokata yrityksen tarpeiden mukaan.

Avainsanat	enegiansäästö, vihreä ICT, tietoverkot

Contents

1 Introduction 1

2 Energy Usage Automation 3

3 Network Devices 5

3.1 Juniper Networks Devices 5
3.2 Cisco Devices 7
3.3 Other Devices 10

4 Auxiliary Technologies 11

4.1 Junos Scripting 11
4.2 Cisco Scripting 17
4.3 Other Related Technologies 21

5 Testing 24

5.1 Testing Plan 24
5.2 Script Execution 25
5.3 Power Measurements 40

6 Energy Saving Calculations 43

7 Discussion 45

8 Summary 47

References 48

Appendixes
Appendix 1. Junos SLAX script
Appendix 2. IOS Tcl script

Glossary

CLI	*Command-line Interface* is a way to interact with a computer by simply typing lines of text as commands.
EEM	*Embedded Event Manager* is a Cisco IOS process to detect networking events and apply automation.
EIGRP	*Enhanced Interior Gateway Routing Protocol* is a proprietary routing protocol from the network device manufacturer Cisco.
IOS	*Internetwork Operating System* is a network operating system used in Cisco networking devices.
NSM	*Network and Security Manager* is centralized network management software to administer various different network devices.
PoE	*Power over Ethernet* is a technique to pass electrical power through Ethernet cabling.
PPS	*Packets per Second* is a way to measure network throughput. It indicates the number of packets that are processed in one second.
RPC	*Remote Procedure Call* is a protocol which is used to invoke a subroutine or a procedure on a remote program in another computer.
RTT	*Round-Trip Time* is the length of time it takes for a signal to be sent and an acknowledgment of that signal to be received.
Tcl	*Tool Command Language* is a scripting language widely used in different embedded systems.
SCP	*Secure Copy* is an SSH based protocol and a program to transfer files securely.

SLAX *Stylesheet Language Alternate Syntax* is a language to write Junos scripts. SLAX is an alternative to the XSLT to make scripts easier to maintain and understand.

SNMP *Simple Network Management Protocol* is a standardized protocol to manage and monitor network devices.

SOAP *Simple Object Access Protocol* is a lightweight protocol used for exchanging structured information between devices or programs in a decentralized, distributed environment.

SSH *Secure Shell* is a cryptographic protocol for secure data communication.

XML *Extensible Markup Language* is a markup language that defines a set of rules for encoding documents to be readable both by a human and computer.

XPath *XPath* is a query language for addressing parts of an XML document.

XSLT *Extensible Stylesheet Language Transformations* is an XML based language to convert XML based documents into other forms.

1 Introduction

In the modern world environmental problems are constantly becoming a bigger issue. As people are craving for new technologies and gadgets, more and more energy is used at the same time. The IT field has grown rapidly for the past decades and only lately this field has started to advance to a more environmentally friendly direction. When it comes to network devices, many people do not really know how much energy these devices consume, how many network devices there actually are or even what they do. Usually the power consumption of the network devices is not even considered when they are bought.

Network device scripts can be used to automate tedious networking tasks in the corporate world. There was a course at Metropolia University of Applied Sciences related to Green ICT where the students had to brainstorm environmentally friendly ideas. The concept of network automation was taken a bit further in this course from only automating configuration tasks to controlling devices and their power consumption automatically. This is where the idea for this thesis started. Basically the idea was just to shut down the devices connected to network equipment when they are not needed, and then power them up again.

There are several devices which can be powered through a network device, but especially IP phones and wireless access points are not needed all the time. These devices could be shut down after office hours to reduce energy consumption. With scripting, these devices can be shut down automatically using different variables. For example at a certain time of the day or when a worker shuts down a computer, the worker's IP phone could be shut down automatically. The approach described in this thesis is to shut down and power up devices automatically at a certain time of the day, because this is easier to deploy into different environments, since no other information is needed than the cable locations of the controlled devices.

From the introduction of the involved network device manufacturers the thesis delves into the more technical details of the implementation. Besides the technical chapter (auxiliary technologies), the thesis is constructed so that it would be easy to approach also for someone without much technical background. The testing chapter is also somewhat technical, but an easily understandable discussion part is included to analyse the results. The outcome of the testing will be further discussed in the subsequent chapters. The energy saving solution is implemented to the devices of two of the biggest enterprise network device manufacturers and practical energy calculations and measurements are presented to show the effects.

The outcome of this thesis is intended to offer something practical that could actually be used to reduce energy consumption in companies. Because saving energy with network devices is not the first thing that comes into someone's mind, practical calculations and tests are made to show how much energy and money could be saved. The latter seems to always be more important in the corporate world.

2 Energy Usage Automation

Nowadays energy usage is controlled automatically in many ways. One good example of automated control of energy usage is to adjust room temperature automatically with a thermostat or switching lights on by using motion sensors. Devices are simply turned off automatically when they are not needed, so why not do this to network devices or devices attached to them? There are many efforts to decrease energy usage of devices, but before that it is more important to consider if they are really needed to be constantly powered on.

There are some recent studies which examine the problem of increasing energy usage of network devices. Most of these studies try to tackle the issue with re-engineering by introducing more energy-efficient technologies for example replacing CPU's with integrated circuits which are designed for a particular use. These circuits consume less energy and are faster, but they lack the flexibility of a CPU which can be programmed to execute any existing computing operation. Optical switching architectures can potentially provide much lower power dissipation than current electronic based network devices, but large scale adaptation of these devices is still far in the future. Some approaches suggest dynamic adaptation to traffic with idle logic and performance scaling, but the drawbacks are delays due to wake-up times and longer packet service times. Some studies come close to the topic of this thesis by suggesting sleeping and standby modes or completely shutting down devices. It seems that when designing more energy efficient network devices, sacrifices in performance would have to be made. This seems to be holding back the development in many areas. [1, 6-9.]

Most of the network devices have been designed to be always on and there is even no proper way to shut them down temporarily. Of course internet core routers cannot just be turned off. There are some network devices which need to be always on, but there are also many devices which are not used all the time. Network devices are rarely used during the night at home or at an office for example. Corporate network devices hog up a huge amount of electricity as it will be demonstrated later in this paper, but some might say that at home energy cannot be saved by turning off network devices. People are advised to unplug mobile phone chargers from the socket when they are not used, so why not turn off network devices as well which use a lot more energy?

Energy saving at home is presented as an idea but this thesis will concentrate on business world solutions. Corporate networks are much bigger and there are also numerous devices that are connected to network equipment, which gives more possibilities to save energy. For example, it is quite common to power IP phones or wireless access points with a network switch using Power over Ethernet (PoE). PoE is a technology used to pass electrical power to devices through Ethernet cabling. Assuming there is an office where there is no one after normal office hours, a huge amount of energy could be saved by shutting down the phones and wireless access points. Unplugging and shutting down devices is considered to be a tedious task and that is why automation is needed. This will be the main idea of this thesis: shutting down PoE devices automatically when they are not needed.

3 Network Devices

3.1 Juniper Networks Devices

Juniper Networks Inc. is an American networking company which was founded in 1996. Juniper started its business with high-performance core routers and quickly advanced to other network devices. Nowadays Juniper manufactures a wide range of different network devices from firewalls to network switches. [2.]

This thesis was intended to be vendor independent when it comes to the network devices, but a decision had to be made to pick specific device manufacturers for the practical testing phase. Juniper Networks devices were chosen, because Juniper is one of the few vendors that have support for XML API and scripting also for their entry level network devices. Juniper is also one of the largest corporate network device manufactures, so their products are used widely all over the world. [3.]

Juniper Networks' product portfolio is very extensive and the main products include routers, switches, VPN and firewall devices, WAN acceleration devices, intrusion prevention systems and wireless networking devices. As Juniper's business grew, it acquired other companies to broaden its device selection. For example, firewall devices were acquired from NetScreen and the first firewall products were denoted by letters SSG in the product name. They used the ScreenOS operating system developed by NetScreen [4]. The new firewall products, which are based on the Junos operating system, are denoted by letters SRX. All the new Juniper devices are based on Junos and they share the same codebase, so new features can be easily deployed to any product line [5, 77]. Here are some of the main product lines and their names:

- EX: Ethernet switches
- IDP: Intrusion Detection and Prevention Appliances
- M: Multiservice Edge Network Routers
- SA: SSL VPN Appliance
- SRX: Firewall products

Junos is a network device operating system developed by Juniper Networks. It is based on FreeBSD, and the Unix shell is accessible within Junos, but the network device management is carried out through the Command-line Interface (CLI). The CLI is accessible via a terminal cable, Telnet or SSH connection. Junos also supports management via XML API, web interface, SNMP and centralized network management platforms (Junos Space or NSM).

All the different Junos devices use the same Junos core, so for example some of the firewall features are accessible from switches. This allows more robust security management in the switches. Users can for example create firewall policies to filter network traffic passing through the switch. In contradiction, firewalls can use advanced routing features which were originally developed for the routers. Deploying Junos devices across a network can also reduce the level of complexity in the network. Instead of mastering multiple network operating systems, the network admins need to learn and maintain just one operating system. This decreases overall complexity of the network and can also reduce training costs. Troubleshooting problems becomes also easier when there is just one operating system and similar practices can be used to troubleshoot all the devices. [6; 7, 8.]

When accessing a Junos device's console with the root user, a Unix shell is opened instead of the Junos CLI. The shell can be used to access the device's file system, and it can be useful for some maintenance tasks and debugging. The other user accounts will start from the Junos CLI, but the shell can also be opened through the CLI. The Junos CLI includes two operating modes: the operational and configuration mode. The device can be managed using the operational mode, and the configuration mode is for making changes to the device configuration. Juniper has a different approach compared to some other vendors when making changes to the configuration. All the changes have to be committed before they become active, so the user can make several changes to the configuration and apply them all simultaneously. Changes made to the configuration in the configuration mode are saved in a candidate configuration before they are committed to the running configuration. This allows for example changing the management IP address and gateway address without the fear of losing connectivity after changing the IP address. [8, 15, 33.]

XML is an essential part of Junos scripting and it is widely used throughout the Junos operating system. The Junos configuration file is XML based and actually all the output in the CLI can also be displayed in XML. This is very good for integration, automation and scripting since it is considerably easier to handle XML than just normal text output when designing a computer program or a script. Also all the commands can be sent in XML format to the Junos system. [5, 109.]

Juniper devices are very flexible when it comes to controlling them. There are various ways to control them and the practical testing part of this paper describes how scripts are used to control the network devices and the devices attached to them. More information about scripting in Juniper devices can be found in the auxiliary technologies chapter.

3.2 Cisco Devices

Cisco Systems, Inc. is a networking company which is, no doubt, known by everyone who is working in the networking field. It is not really a big surprise since Cisco has the biggest market share in enterprise networking as it can be seen in figure 1. Everything started when Cisco was founded in 1984 by Len Bosak and his wife Sandy Lerner. They were working at Stanford University and they had an idea to build and sell Internet routers to research and academic institutions. A long time has passed since that time and Cisco is now a networking giant manufacturing a wide range of different network devices. Apart from the core networking, Cisco also has a hold on the video conferencing and internet telephony market. [9, 359.]

Figure 1 shows Cisco's market share and other related factors along with the competing vendors. Juniper and Cisco, the manufacturers this paper deals with, are positioned among the top three manufacturers in the chart. Brocade, which holds the second place, is more known for its data center solutions rather than Ethernet networking [10]. It would not be the ideal example for this paper, because data center services are designed to be always available and energy automation would be problematic to implement.

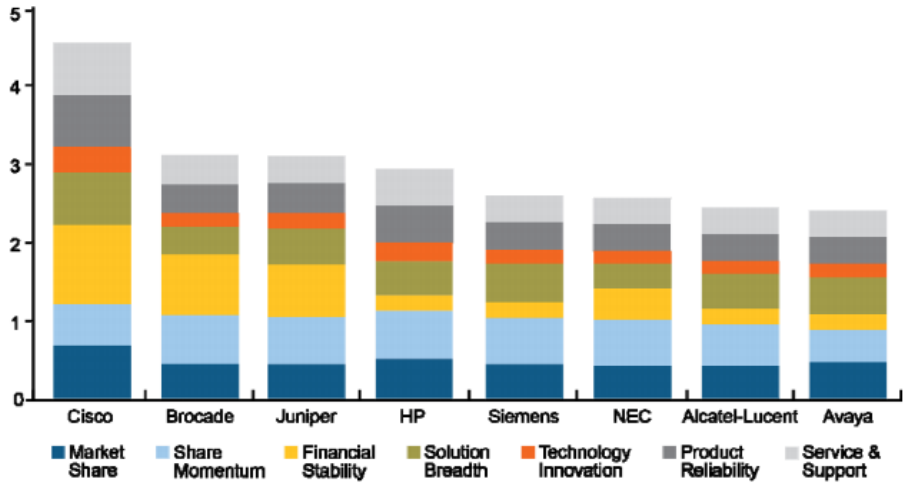

Figure 1. Top networking and communication equipment vendors in 2012 [11, 4]

Cisco devices were included in testing, because Cisco also has a very powerful scripting support like Juniper, though it lacks XML API. XML API can only be found in high-end carrier-grade routers [12, 9]. Cisco is also a very good test subject as it holds the biggest market share in enterprise network devices and is widely known by network technicians and companies. Cisco and Juniper devices are also easily accessible at Metropolia University of Applied Sciences where all the devices used in this thesis were obtained.

IOS is the primary operating system for the majority of Cisco's devices. For example Cisco's Nexus series switches use a different operating system. Unlike Juniper's Junos operating system, Cisco's IOS is extremely fragmented. There are several versions of IOS for each Cisco hardware product with numerous releases. Determining what features are in each release and which release should be run on each device can be a time consuming task. [5, 73, 82.]

IOS CLI has many operating modes but they can be roughly divided into EXEC and configuration mode. The EXEC mode is mostly used to get runtime information from the device and the configuration mode is used to modify the device configuration. The EXEC mode is further divided in to two different modes: User EXEC mode and privileged EXEC mode. The user EXEC mode can be used to adjust terminal settings, perform basic tests and view system information. From the privileged EXEC mode the user can also configure operating parameters which affect the operation of the device

and it can be protected with an additional authentication. The configuration mode has different levels depending on what the user is configuring. For example, the global configuration mode is used to change global settings which affect the whole device and the interface configuration mode is used to change settings of an individual interface. All the configuration commands in the configuration mode are effective immediately when they are typed, so multiple changes cannot be made at the same time like with Junos. There is also a mode called ROM monitor mode for recovery, when the Cisco IOS software cannot be loaded properly.

There are multiple ways to access IOS devices but the CLI is usually the best way to configure IOS switches and routers. CLI is accessible via a console cable, Telnet or SSH connection. SSH of course should always be preferred to Telnet, since it is an encrypted protocol. Most of the devices also have a web interface, but the web interface can be very limited in some devices. Cisco ASA firewalls have a Java based management tool called ASDM, which can come in handy for example when configuring large access tables. [13, 39.] There is also a GUI based configuration tool for Cisco access routers called Cisco Configuration Professional to simplify configuration through GUI-based wizards. [14, 1.]

Cisco has been the market leader for long while other manufacturers have tried to take their slice of the network device market. Changing a Cisco environment to another manufacturer might not be so straightforward, since Cisco has introduced many proprietary protocols, which do not work with the devices from other manufacturers. Sometimes it is faster to implement one's own technologies than wait for a long standardization process to finish, but this also creates compatibility issues with other vendors. A good example of a fully proprietary protocol is the routing protocol EIGRP. This routing protocol works only in IOS devices and other manufacturers were not able to implement EIGRP in their own devices (Cisco released EIGRP later as an open standard [15]). Juniper Networks has a different approach and uses standardized protocols. This guarantees that the devices have less compatibility problems with devices from other manufacturers.

Cisco devices are a very interesting test subject because of their popularity in the networking field. If energy can be saved with the existing devices just by invoking a simple script inside the device, it will be a very affordable solution to save money and energy for many companies who use Cisco devices. The testing part of this paper describes

how scripting was used in Cisco devices to see how it can be utilized for energy saving in different situations. Although XML API is not supported in Cisco devices, there are several ways to execute a script outside the device. More information about scripting and script execution can be found in the auxiliary technologies chapter.

3.3 Other Devices

By no means are Juniper and Cisco the only manufacturers whose devices can be used for energy usage automation. Juniper and Cisco had the needed technologies to implement the solution described in this thesis without difficulties, but there are many other network devices with similar capabilities and features. In addition, devices do not necessarily need to support scripting. Scripts can also be external as long as there is a way to send the needed commands to the network device through the network.

Usually network devices support at least Telnet and SNMP. These both can be used to control the device automatically, but the Telnet interface is primarily designed for human interaction so it is not the ideal way. Telnet also poses a security risk as it is not an encrypted protocol, but in this case SSH could be used if it is supported. Another problem would be proper error handling, which should be implemented by the programmer to the external script that connects to the device. Nevertheless, it is possible to use Telnet or SSH, but it requires a little more work and there might be unexpected problems.

SNMP is supported by many network devices and it is designed to manage and monitor them, so it would make a good option to implement energy automation. Although SNMP does not provide the same flexibility as scripting inside the network device, operations like shutting down ports could be executed very easily. SNMP includes a command to control and modify network device parameters. The first version of the SNMP protocol did not support any encryption, so controlling commands were rarely used and SNMP was mainly used for monitoring. The newest version supports encryption and also has other advanced security features like playback protection. Playback protection ensures that a received message is not a replay of an earlier message. [9, 806-820.]

4 Auxiliary Technologies

4.1 Junos Scripting

The Junos operating system supports scripting and users can make their own scripts to automate the device operation. Scripting is supported throughout the whole Junos based device portfolio including entry level devices. Junos scripting is XML based and can be written in XSLT, or in a little more user friendly way, in SLAX. Scripting does not only save time when automating tedious and time consuming tasks, but it can also make a network more resilient when it can automatically react to errors and possibly even correct them.

XML

Junos scripting is XML based so in order to understand Junos scripting it is good to review the basics of XML first. World Wide Web Consortium (W3C) defines XML as follows: “Extensible Markup Language, abbreviated XML, describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them.” [16]. Junos automation scripts communicate with their host device using the XML language. Junos XML API provides procedures for Junos scripts to make requests and these requests can instruct other processes to retrieve data or perform specific operations. After requested operations are performed, Junos returns XML results to the script engine for further analysis. A script can for example request a part of the configuration from the XML API. Listing 1 below shows an example of an XML reply from the XML API.

```
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/9.6I0/junos">
  <configuration junos:commit-seconds="1238100702" ju-
nos:commitlocaltime="
2009-03-26 13:51:42 PDT" junos:commit-user="user">
    <routing-options>
      <route-distinguisher-id>192.168.1.1</route-distinguisher-id>
      <autonomous-system>
        <as-number>65535</as-number>
      </autonomous-system>
    </routing-options>
  </configuration>
  <cli>
    <banner></banner>
  </cli>
</rpc-reply>
```

Listing 1. XML reply from the Junos XML API [7, 12]

In the XML code, “rpc-reply” in the first line indicates that it is a reply from Junos providing the requested XML data. The next line indicates that the reply is configuration information, and the configuration data starts from the following line. To understand the example above, it is necessary to explain a few key concepts of XML: elements, attributes and namespaces.

XML code consists of hierarchical *elements* which are the basic unit of information in an XML document. Elements can contain other elements or data such as a text string or a number. An element enclosed by another element is called a child element and the upper element is called a parent element. For example, in the XML reply above, “routing-options” is the parent element of “autonomous-system” and a child of the “configuration” element. In order to be hierarchical, all elements have a start and tags which set the boundaries for the element. All the element data is placed between the starting and ending tags. Tags are enclosed within < > signs. In the XML reply above “<routing-options>” is the starting tag for routing configuration and the ending tag is “</routing-options>”. Both tags include the element name and the ending tag includes a slash before the name. If the element is empty and does not have any child elements, it can be expressed by just one tag which ends with a slash.

Attributes can provide additional information to the elements. Attributes are inside the starting tag of the element after the element’s name. They are expressed by an attribute name followed by an equal sign. Attribute data is placed inside quotation marks after the equal sign. An element can contain multiple attributes and they are separated by a space. For example, in the XML reply above, elements “rpc-reply” and “configuration” both contain attributes. In the listing one can also see that the attributes defined for the “configuration” element all start with the same word “junos”. This word has a special purpose as it indicates the *namespace* of the attribute. Namespaces prevent confusion between different attributes which have the same name for different purposes. Multiple devices and systems can use the same attribute name, but when a namespace is included, it is explicitly known what the attribute indicates. [7, 12-14.]

XSLT and SLAX

Since the Junos script engine uses XML to communicate with other Junos processes, also the scripts are written in XML, more precisely in XSLT. Extensible Stylesheet Language Transformations is a standard to convert XML documents into other XML documents or objects. XSLT works like Cascading Style Sheets (CSS) with HTML; it does not modify the original document, but rather supplements it.

XSLT includes programming instructions such as “if” and “for” statements, but it can be quite complicated to write scripts in XSLT for someone who is not accustomed to it already. That is why Juniper developed SLAX, Stylesheet Language Alternative syntaX. SLAX does not add anything to the XSLT engine; it just makes it simpler to use. SLAX has a similar syntax to C and Perl programming languages, so it is easier to learn for someone with previous programming experience. SLAX is designed to have more readable syntax than XSLT and the configuration in the SLAX scripts can be displayed similarly to the way the Junos CLI shows it. [7, 11.] Ultimately all SLAX scripts are converted to XSLT and passed to the script engine as XSLT, so it does not really matter in which form the scripts are written [17]. Figure 2 clarifies the relationship between SLAX and XSLT:

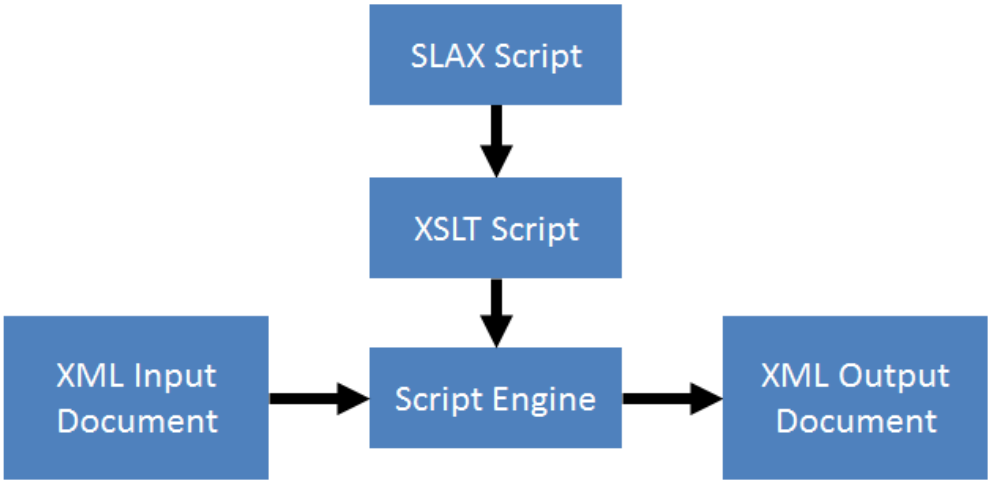

Figure 2. Processing a SLAX script [17]

There are a few key differences in SLAX which make it easier to read and type. First, if an element contains child elements, then the child elements are contained within curly braces, so no ending tags are needed. Second, data inside an element is written inside quotation marks and the line is terminated with a semi-colon. Lastly, empty elements

with no children are represented by a single start tag followed by a semicolon. These differences do not make the code only easier to read but also more similar to the Junos configuration and programming languages like C and Perl. [7, 16.] Listings 2 and 3 demonstrate the differences between XML and SLAX.

```
<interfaces>
   <interface>
     <name>ge-0/0/0</name>
     <disable/>
   </interface>
</interfaces>
```

Listing 2. Junos configuration XML data structure [7, 15-16]

```
<interfaces> {
   <interface> {
      <name> "ge-0/0/0";
      <disable>;
   }
}
```

Listing 3. Junos configuration in SLAX format [7, 16]

There are three different types of Junos scripts identified by the way they are called and executed in the system: commit scripts, event scripts and op scripts.

Configuration automation scripts (commit scripts) are executed during the commit process. These scripts can be used to avert human made errors or automate configuration changes. Together with event scripts, a commit script could be used for activating and deactivating a firewall filter rule based on configured time ranges. Another possibility could be to confirm that the configuration is consistent between the routing protocol and interfaces or just a script which checks for typical human made errors in the configuration.

Event Automation scripts (event scripts) are executed in response to an event that occurs in the system. With event scripts, networking devices can be self-monitoring and self-diagnostic. An event script could for example ping a remote host and write RTT details to a log file or automatically switch between a primary and alternate next hop based on a ping result. RTT tells the time how long it takes for a ping packet to travel to the destination and back, so it is a good way to monitor network operation. Event automation scripts are also just the right scripts to be used for energy automation by automatically disabling power consuming features like PoE ports based on different variables. Event scripts were used in the practical testing part of this project.

Operations Automation scripts (op scripts) are executed by the user from the command line or when called by another script. Op scripts can automate manual tasks that would normally need user interaction and several different commands, and therefore they can be used to improve the operational efficiency. There can be countless of different op scripts. Some can for example display information like CPU usage or even connect to remote hosts to execute commands. Op scripts can be very useful when automating tedious tasks that would normally need several commands to be typed by the user. [7, 8-10.]

Nearly everything in the Junos system can be automated with the Junos scripts. Scripts can be executed manually, after a certain time period or after a certain event. A script engine can use the commands defined in the XML API to communicate with the device and other processes. It is easy to view the XML hierarchy within Junos since all the output of CLI commands can be viewed also in XML format. Below in listing 4 one can see how routing configuration can be seen in XML format from CLI.

```
user@Junos> show configuration routing-options | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/9.6I0/junos">
  <configuration junos:commit-seconds="1238100702" ju-
nos:commitlocaltime="2009-03-26 13:51:42 PDT" junos:commit-
user="user">
    <routing-options>
      <route-distinguisher-id>192.168.1.1</route-distinguisher-id>
      <autonomous-system>
        <as-number>65535</as-number>
      </autonomous-system>
    </routing-options>
  </configuration>
  <cli>
    <banner></banner>
  </cli>
</rpc-reply>
```

Listing 4. Routing configuration in XML format [7, 12]

A deeper look into the scripting itself is given next. The best way to get acquainted with a new programming language is to see the actual code. Listing 5 shows an example of a “Hello World” script written in SLAX:

```
version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";

match / {
   <op-script-results> {
      <output> "Hello World!";
   }
}
```

Listing 5. Hello World written in SLAX [7, 18]

The first line "*version 1.0;*" is a required line in the beginning of all Junos scripts. It indicates the version of the SLAX language. Currently the newest version of SLAX is 1.1 [18]. The next three lines starting with *ns* define a namespace prefix and its associated namespace URL. These three namespaces must be included in every Junos script. In the next line, an *import* statement is used to import code from another script. Junos.xls script contains useful default templates and parameters, so all scripts should import this file. After the import statement there is the main template for op scripts: "*match"*. The script engine always starts code execution from the main template and all code must be included within code structures called templates. [7, 22-23.] Listing 6 below shows the output generated by the Hello World script when it is executed in the CLI:

```
user@Junos> op hello-world
Hello World!
```

Listing 6. Hello World script executed in console [7, 18]

Although the syntax is similar to C and Perl, understanding how SLAX and Junos scripting works might require some time. For example, before version 1.1 of SLAX, all the variables could only be declared once and after that they could not be changed anymore. SLAX has a very different approach to programming and understanding XML and related technologies is very helpful. Sending, receiving and parsing XML data within the script is an essential part of Junos scripting.

The Junos scripting system is a very powerful tool to automate system tasks and to increase administrative efficiency. All Junos processes communicate with each other using XML and even external applications can connect to the XML API and execute commands and scripts. This creates a possibility to retrieve and display information using custom end-user interfaces. When considering energy automation, a web interface could be used to monitor power consumption and device status in addition to the

scripts. Sometimes automation needs to be modified, so users could for example modify the time when network devices shut down or disable automatic scripts from the web interface. Shutting down devices manually could also be possible. Junos scripting is a very extensive topic and for those who want to know more reading the book *This Week: Applying Junos Automation* by Curtis Call [7] is much recommended.

4.2 Cisco Scripting

Cisco IOS scripting provides similar capabilities for Cisco devices as the Junos scripting for Juniper devices. Commonly scripts are used for monitoring, troubleshooting and to increase work efficiency by automating time consuming tasks. Cisco devices support Tool Command Language (Tcl) to create scripts and Embedded Event Manager is used to execute the scripts based on different triggers and variables.

Tool Command Language

Tcl was invented in the late 1980s by John K. Ousterhout. It is a dynamic programming or scripting language, an interpreter, and a C library. Tcl is widely used apart from Cisco devices and it is considered a fairly simple scripting language. Some examples of the usage of Tcl are testing and automation, web applications, desktop GUI applications, databases and embedded systems. Tcl is an interpreted programming language, so it does not need compiling, which makes it good for scripting purposes. An interpreted language is not the best performance wise, but the development process is much faster compared to compiled languages and scripts can be modified easily. Performance is not usually an issue when making simple scripts, but carrying out small changes in the scripts is common, so it is very helpful that these changes can be made quickly. [19, 1-3.]

Compared to Juniper's SLAX, Tcl syntax seems much simpler. Listing 7 below shows a hello world script written in Tcl.

```
puts "Hello World!"
```

Listing 7. Hello world in Tcl

Just one line is needed for the hello world script as it can be seen from the example above. This is a huge difference compared to SLAX. Tcl consists of commands separated by new lines or semicolons and the only variable data type is a string. Tcl syntax is quite fast to learn for someone with previous programming experience.

Embedded Event Manager

The Tcl shell was first introduced in the modular Catalyst 6500 series switches, but later also in the access layer switching devices with IOS version 12.2(40)SE. Tcl scripts can be written and executed directly inside the Tcl shell in Cisco devices, but manual execution is not really the optimal way to use scripts. That is when Embedded Event Manager (EEM) comes along. EEM is an IOS process which monitors software and hardware components with event detectors and it can trigger a desired automation after a certain event is detected.

EEM is a very powerful tool and it monitors several components in the networking device. After an event is detected a policy can be defined to take desired actions. Policies are scripts which can be applets or Tcl scripts. Applets are simple scripts which consist of regular IOS command-line interface commands, so the user does not need to be familiar with scripting. Only basic knowledge of Cisco IOS commands is sufficient to create applets. Tcl scripts can also use Cisco IOS commands but they are not constrained to use them and therefore can be used much more flexibly to create advanced automation scripts. EEM continues to evolve and as for now there are over 20 different event detectors to trigger scripts. In listing 8 below, there are some of the event detectors to demonstrate the possibilities of EEM and script execution. [19, 3-4.]

- `Syslog`
- `SNMP`
- `Timer`
- `IP SLA`
- `Interface`
- `CLI`
- `NetFlow`
- `Routing`
- `Remote-procedure call (RPC)`

Listing 8. Examples of EEM event detectors [19, 57-59]

This is not a complete list of the event detectors, but as it can be seen from the list above, EEM is very versatile and automation can be used in various different parts of the system. EEM consists of three main components: event detectors, policies and the EEM server which binds these together. The EEM server is the heart of EEM and it works as a bridge between policies and internal Cisco IOS subsystems used by the event detectors. The EEM server has many functions: registering events which occur in the IOS subsystems, storing information about events, publishing events, requesting additional information about events, registering internal script directories, registering applets and Tcl scripts and executing user-defined scripts. Figure 3 below clarifies the structure of the Embedded Event Manager. [19, 56.]

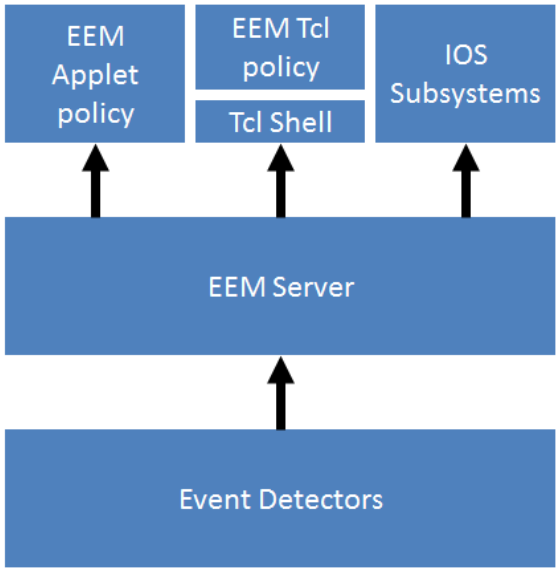

Figure 3. Embedded Event Manager structure [19, 5]

EEM and Tcl hold some quite interesting technologies considering energy automation. From the version 2.4, EEM supports Remote Procedure Call to execute scripts remotely [19, 179]. There is also an Embedded Menu Manager (EMM) which can be used to easily create menus for scripts with XML [19, 139]. Another interesting feature is that Tcl can be used to create a web server and scripts can be executed through a customized web interface [19, 144]. For device and energy management a centralized management solution would be better, rather than a web server inside every network device, but it is certainly an interesting feature and worth mentioning. RPC support on the other hand could be used for a centralized solution. RPC requests can be sent via an SSH connection using Simple Object Access Protocol (SOAP) to encapsulate the mes-

sage. An RPC request can invoke a script and the network device provides an answer to the requester. RPC could be used to invoke scripts remotely from a centralized management server, for example. [19, 179.]

Basically Juniper and Cisco devices provide the same functionality but with different technologies. Cisco relies on Tcl scripting which is widely used in other systems too and can be considered a programming language rather than just a scripting language. Juniper on the other hand uses a proprietary scripting language, although it is XML based which is an open standard. Creating a web interface to control Cisco devices can be implemented using technologies like SSH, SNMP, RPC or their combination, but Juniper offers an XML API for remote control which greatly simplifies the implementation. Juniper also includes scripting and XML API in all Junos devices, while Cisco scripting and EEM event detector support vary throughout their product portfolio.

Cisco scripting enables a wide range of customizations for Cisco networking devices. It can be used for automation, monitoring, error recovery and much more if needed. Realizing the potential of scripting can reduce other than energy costs too. For example enterprise network management software can be very expensive and scripting could replace them in some smaller companies. The testing chapter describes how Cisco scripting is used to shut down connected devices. This thesis only covers the basics of Tcl and Cisco scripting. To find more information about Tcl, the reader is encouraged to visit the website http://www.tcl.tk/. A book called *Tcl Scripting for Cisco IOS* [19] covers Cisco scripting and its capabilities very extensively and it is highly recommended to anyone who wants to learn more about this subject.

4.3 Other Related Technologies

SSH

Secure Shell (SSH) is a cryptographic network protocol for secure remote login and various other secure network services over an insecure network. It is widely used in networking devices to provide a secure remote connection to the CLI, but it can also be used for much more. The SSH connection protocol provides channels which can be used for many different purposes. Standard methods include forwarding arbitrary TCP/IP ports through the SSH connection which enables encapsulating and encrypting basically any TCP connection.

SSH consists of three major components: transport layer protocol, user authentication protocol and connection protocol. The transport layer protocol provides authentication, confidentiality, integrity and optional compression. The user authentication protocol is used to authenticate the client-side user to the server. The connection protocol multiplexes the secure SSH tunnel into several logical channels and it runs over the user authentication protocol. [20.]

All of the enterprise network devices do not support scripting but almost all of them can be remotely accessed with SSH. SSH is used to access the CLI remotely and it provides the same functionality as Telnet but SSH is encrypted. Automation could be implemented with a remote script which connects to a network device with an SSH connection and then executes commands. CLI is designed for user interaction, so it is not the ideal way to control the device with a remote script which inputs commands to the CLI. There could be difficulties for example with error handling. However, SSH could be used to send commands and shut down ports or devices and reduce energy consumption on devices without scripting support.

SNMP

Simple Network Management Protocol (SNMP) was first introduced in 1993 and has now evolved from the first version to SNMPv3. SNMP can be used to retrieve information from networking devices and to control them. Information can be polled from the network devices or the network device can send a trap message invoked by a networking event to the monitoring entity. The first version of the protocol is not secure but later versions, especially SNMPv3, have included many security features like encryption and access control. Because of the lack of security features, SNMPv1 was mainly used for monitoring rather than controlling. SNMPv3 was released in 1999 and updated in 2002, but still some devices lack the support for the latest version of SNMP standard, though the biggest manufacturers have updated their devices to the support latest version during recent years. SNMP can be a very powerful tool for network control if security is handled properly.

SNMP is a framework for network management which defines security and administration capabilities, management information objects, data definition language to format the information being exchanged and a protocol to exchange information and commands between a managing entity and an agent. SNMPv2 defines seven types of messages to be exchanged between a managing entity and monitored device. Listing 9 shows the different message types with their senders and receivers. [9, 806-817.]

```
Message Type            Sender-Receiver
GetRequest              manager-to-agent
GetNextRequest          manager-to-agent
GetBulkRequest          manager-to-agent
InformRequest           manager-to-manager
SetRequest              manager-to-agent
Response                agent-to-manager or manager-to-manager
SNMPv2-Trap             agent-to-manager
```

Listing 9. SNMPv2 message types [9, 816]

The most common message types from a managing entity to a monitored agent are *GetRequest* and *SetRequest.* The “get” request is used to get a value from one or more objects. It can be used for example to get network interface statistics or system session counters. The “set” request is an important message type for remote controlling as it can be used to modify objects. The “set” request could for example be used to shut down PoE ports, if this kind of request is supported in the device. “agent-to-manager” messages include *Response,* which is self-explanatory, and *SNMPv2-Trap*.

SNMP traps can be used to send information to notify a manager when something exceptional happens like a restart of the device, an interface changing its status or a link reaching a predefined congestion limit. [9, 814-816.]

This is a very brief and simplified introduction to SNMP, but the protocol is in no means simple anymore unlike its name suggests. The SNMP protocol has evolved enormously and there are dozens of standardization documents to define it. SNMP sounds like an ideal management protocol with the advanced security features in the latest version for implementing energy automation, but there are several advantages using scripting instead of SNMP. Although SNMP is supported by majority of the network devices, support for the SNMPv3 with enhanced security features is not always included. Scripting is a built-in feature which can access the network device regardless of external factors like network connection. There are numerous monitoring programs to utilize SNMP, but these programs can be quite complex and take a long time to deploy. Commercial SNMP monitoring programs can also be quite costly. Scripts on the other hand are easy and fast to deploy and do not require any additional software. Also security is not such a big concern, because no external connection is needed.

5 Testing

5.1 Testing Plan

Practical testing with actual scripts is dealt with in this chapter. The output after possible corrections should be two working scripts which can be used for energy automation. All the scripts described in this chapter can be found in the appendixes. Some power measurements are also analysed in this chapter and the following chapter will examine the power consumption in an example office and discuss what kind of impact these scripts can have to the energy consumption.

First, a Juniper network switch was tested with a script which is available for download from the Juniper website. This is a simple script to shut down and enable PoE ports at a certain time of the day. In the thesis a Cisco network switch was tested with a custom script which has almost the same functionality as the script for Juniper devices. All the tests are examined in this chapter and making further modifications to the scripts is also discussed briefly.

Table 1. Test equipment

Juniper switch	EX3200-24T
Cisco switch	WS-C3560V2-24PS-E
IP-phones	Cisco IP Phone 7960 Cisco IP Phone 7912
Wireless access points	Cisco AIR-AP1231G-E-K9 Cisco AIR-AP1230A-A-K9

Four PoE devices were connected to the network switches for testing. These devices are two IP-phones and two wireless access points. These are very common PoE devices and can be found in many companies. Scripts are supposed to turn off these devices and then power them up again. It is important to make sure that the scripts are executed at the right time and by the right triggering event. Scripts should not affect other functionality or shut down wrong ports. A list of the used test equipment can be seen in table 1 above.

5.2 Script Execution

Energy Automation with Junos Scripts

An example script to shut down and enable PoE ports called PoE scheduler (poesched) is available on the Juniper website [21]. This script is made by Patricio Giecco for EX-series Junos switches. A switch with a model number EX3200-24T was used to test this script. EX3200-24T is a business grade switch with 24 copper interfaces and two optical fiber interfaces. Only eight of the copper interfaces include PoE support so the device is not a full PoE switch. Nevertheless, eight ports are more than enough for testing the script functionality. At the time of the testing, the switch was loaded with the Junos software version 12.3R6.6.

The script's type is event script. Event scripts are triggered by a certain event in the system. Junos system daemons create events and pass them to the event processing daemon. The event processing daemon then checks event policy configuration if there are any scripts which need to be executed by those events. If the event matches the configuration, the designated script is executed. Figure 4 below illustrates the execution of event scripts.

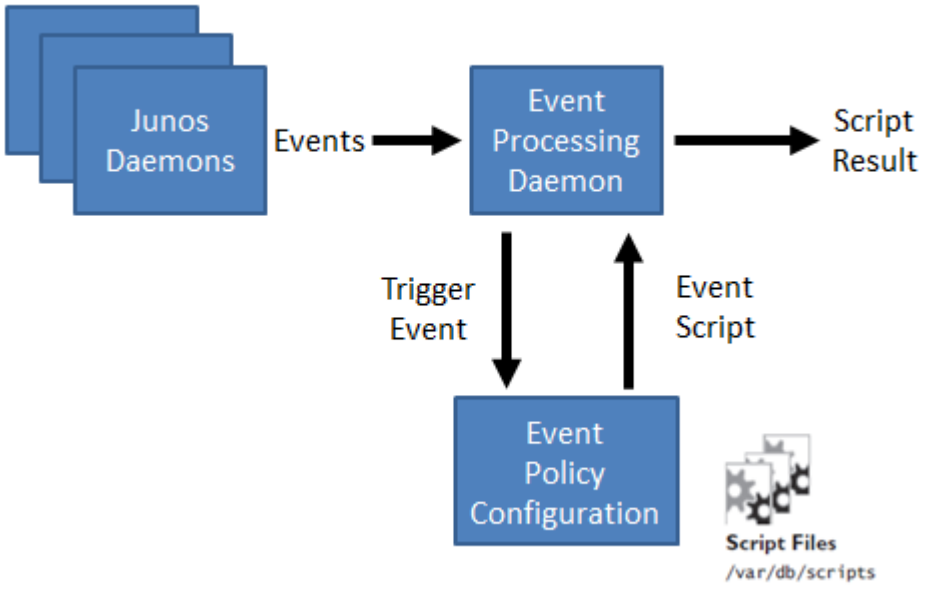

Figure 4. The flow of event processing [7, 81]

By default the script is executed once an hour. It then checks the configuration for schedule configurations from the device configuration. Multiple schedule groups can be made and attached to different interfaces. This enables for example shutting down IP-phones for different departments in a company at different times.

The script starts with namespace declarations and code imports essential to any SLAX script. After this there is a part which is specific to event scripts: event policy configuration which defines the rules how and when the event script is triggered. The event policy configuration for the PoE scheduler script is shown in listing 10 below.

```
var $event-definition = {
    <event-options> {
        <generate-event> {
            <name> 'poe-scheduler-start';
            <time-interval> '3600';
        }
        <policy> {
            <name> 'poe-scheduler';
            <events> 'poe-scheduler-start';
            <then> {
                <event-script> {
                    <name> 'poesched.slax';
                }
            }
        }
    }
}
```

Listing 10. Event policy configuration for poesched script [21]

The policy configuration has a “generate-event” element. This element generates an event called “poe-scheduler-start” once per hour. Below that, there is a “policy” element which defines the “poe-scheduler-start” as the triggering event for the “poesched.slax” script. The event policy configuration can also be defined in the device configuration, but from the Junos version 9.0 it has been possible to embed the policy configuration to the script [7, 114]. This makes scripts easier to deploy and it also reduces the device configuration size.

Next in the code there is a “$arguments” variable. Junos uses this to create help text in the CLI for command line arguments. This is not needed for event scripts since they are not executed from the command line. Command line arguments are defined next after the “param” command. Arguments cannot be passed from the command line for an event script but this can be used to define arguments which can be passed in the event policy configuration to the script. The PoE scheduler has a “time” variable as an argument. Normally this variable is used to store the current system time, but it can

also be defined in the configuration for testing purposes. The actual script starts after argument definitions. The script is enclosed inside the “match” main template. First the script retrieves the current PoE configuration. Listing 11 shows the code to retrieve the configuration.

```
var $poe-rpc = <get-configuration> {
    <configuration> {
        <poe>;
    }
}
```

Listing 11. Retrieving PoE configuration [21]

This code has to be passed to the system for processing. The next few lines open a connection to the script daemon (mgd). These lines can be seen in listing 12 below.

```
var $poe = jcs:invoke($poe-rpc);
var $con = jcs:open();

if (not($con)) {
        <xnm:error> {
            <message> "Not able to connect with local mgd";
        }
    }
```

Listing 12. Connecting to the script daemon [21]

Functions from the “jcs” namespace are used to open the connection. The PoE configuration is saved in the “$poe” variable. Next, any configuration changes are saved in the “$xml” variable. The script contains two nested “for-each” statements to loop through the XML based configuration using XPath. XPath is a query language for addressing parts of an XML document. The code in the first “for-each” statement retrieves the scheduler configuration and saves the scheduler group name and time values in variables. The next “for-each” statement loops through the PoE interface configuration looking for interfaces configured with the previously retrieved scheduler groups. If an interface match is found, configuration action is written into the “$xml” variable. “for-each” statements and the code inside them are in listing 13.

```
for-each ($poe//apply-macro[starts-with(name, "scheduler-") &&
  data/value == number($time)]) {
  var $splitName = jcs:regex("scheduler-(.+)", name);
  var $group = $splitName[2];
  var $action = data[value == number($time)]/name;
```

```
  for-each ($poe//interface[apply-macro/name == "scheduler" &&
    apply-macro/data/name == "group" &&
    apply-macro/data/value == $group]) {
    if ($action == "on" && disable) {
      <interface> {
        <name> name;
        <disable delete = "delete">;
    }
    } else if ($action == "off" && not(disable)) {
      <interface> {
        <name> name;
        <disable>;
      }
    }
  }
}
```

Listing 13. Parsing XML configuration with for-each statements [21]

The code in the last “if” statement of the script commits the configuration if any interface configuration changes are written in to the “$xml” variable. A function from “jcs” namespace is again used to save the configuration. The code to commit the configuration is shown in listing 14. This is the last part of the script and the whole script can be found in the appendixes of this thesis.

```
if ($xml/configuration/poe/interface) {
    var $results = {
        call jcs:load-configuration($connection = $con,
                                    $configuration = $xml);
    }
}
```

Listing 14. Committing configuration [21]

Testing the script

First the script needed to be transferred to the device and be enabled. Scripts can be transferred to the device in many ways, but SCP is a very safe way to transfer the script since it is an encrypted protocol. SCP is an SSH based protocol to transfer files securely. Transferring the script from a Unix shell using SCP is shown in listing 15. All event scripts in Junos have to be placed in the folder “/var/db/scripts/event/”.

```
[user@shell ~]$ scp poesched.slax
root@10.10.10.1:/var/db/scripts/event
root@10.10.10.1's password:
poesched.slax                                 100% 4088
4.0KB/s   00:00
[user@shell ~]$
```

Listing 15. Transferring a script to a Juniper device using Secure Copy (SCP)

After transferring the script, it needed to be enabled. Super-user privileges are required to enable a script from the command line. Scripts are enabled from the configuration mode. Listing 16 shows the steps to enable an event script.

```
root> configure
Entering configuration mode

[edit]
root# edit event-options event-script
root# set file poesched.slax
root# commit
commit complete
```

Listing 16. Enabling an event script in Junos

Interfaces from “ge-0/0/0” to “ge-0/0/3” were used for testing. IP-phones were connected to the ports “ge-0/0/0” and “ge-0/0/1” and wireless access points to the ports “ge-0/0/2” and “ge-0/0/3”. The interface list after connecting all the devices can be seen in listing 17. The list shows that all the interfaces are enabled and status is “up”.

```
root> show interfaces terse
Interface               Admin Link Proto    Local
Remote
ge-0/0/0                up    up
ge-0/0/0.0              up    up   eth-switch
ge-0/0/1                up    up
ge-0/0/1.0              up    up   eth-switch
ge-0/0/2                up    up
ge-0/0/2.0              up    up   eth-switch
ge-0/0/3                up    up
ge-0/0/3.0              up    up   eth-switch
...
```

Listing 17. Interface list before SLAX script execution

The script already includes the event policy configuration with a timer to run it once an hour, but scheduler configuration is needed to instruct the script to shut down correct ports at the right time. Schedulers are saved under “poe” configuration with “apply-macro” commands. Multiple schedulers can be made by writing the scheduler group name after the “scheduler” word. The commands “on” and “off” define at which hour of the day the ports are enabled and disabled. After scheduler groups are created, they have to be linked to the PoE interfaces. Listing 18 shows two different groups and how The “admin” group was enabled in the interface “ge-0/0/0”. The “ge-0/0/0” port was scheduled to be disabled at 18:00 and enabled at 4:00 am. The same schedule was applied to the other interfaces as well when the script was tested.

```
poe {
            apply-macro scheduler-admin {
                        off 18;
                        on 4;
            }
            apply-macro scheduler-finance {
                        off 20;
                        on 9;
            }
            interface ge-0/0/0 {
                        apply-macro scheduler {
                        group admin;
                        }
            }
            ...
}
```

Listing 18. Script scheduler group configuration

The script was enabled and triggered every hour by the timer event. There is a tool in Junos to generate events, so there is no need to wait for the timer to be triggered once an hour. Events have to be generated from the Junos shell with a command “logger”. The logger command is simple to use and only one parameter is needed to generate an event. The event name is given as a parameter and it is important that it is written in upper case. A command to generate the “poe-scheduler-start” event is shown in listing 19.

```
root> start shell
% logger -e POE-SCHEDULER-START
```

Listing 19. Generating an event from Junos shell

After generating the event, the script should have been executed automatically. The execution of the script can be verified by viewing log files. The scheduler disable time was set to “18” and it was after 18:00 when the the “poe-scheduler-start” event was generated. This means that all four PoE ports that were tested should have been disabled. Listing 20 shows the system time and event script log after the script was executed.

```
root> show system uptime
Current time: 2014-06-22 18:00:57 UTC
...

root> show log escript.log | last
...
Jun 22 18:00:46 event script processing begins
Jun 22 18:00:46 reading event details
Jun 22 18:00:46 testing event details
Jun 22 18:00:46 running event script 'poesched.slax'
Jun 22 18:00:46 opening event script
'/var/db/scripts/event/poesched.slax'
```

```
Jun 22 18:00:46 reading event script 'poesched.slax'
Jun 22 18:00:58 event script output
Jun 22 18:00:58 begin dump
<?xml version="1.0"?>
Jun 22 18:00:58 end dump
Jun 22 18:00:58 inspecting event output 'poesched.slax'
Jun 22 18:00:58 finished event script 'poesched.slax'
Jun 22 18:00:58 event script processing ends
```

Listing 20. System time and event script log after first script execution

The log shows that the event script was executed without any error messages. If the script worked correctly, all four ports from “ge-0/0/0” to “ge-0/0/3” should have been down. This can be checked from the interface list. Listing 21 shows what the interface list looked like after the script was executed.

```
root> show interfaces terse
Interface               Admin Link Proto    Local
Remote
ge-0/0/0                up    down
ge-0/0/0.0              up    down eth-switch
ge-0/0/1                up    down
ge-0/0/1.0              up    down eth-switch
ge-0/0/2                up    down
ge-0/0/2.0              up    down eth-switch
ge-0/0/3                up    down
ge-0/0/3.0              up    down eth-switch
...
```

Listing 21. Interface list after SLAX script execution

The interfaces were down so the script worked correctly. Finally the script had to be tested to verify that the interfaces are enabled correctly. Listing 22 includes all the steps from changing the schedule configuration to enable interfaces, executing script and checking the results.

```
root> show configuration poe apply-macro scheduler-admin
off 4;
on 18;

root> show system uptime
Current time: 2014-06-22 18:06:23 UTC
...

root> start shell
root@:RE:0% logger -e POE-SCHEDULER-START
root@:RE:0% cli

root> show log escript.log | last
Jun 22 18:06:39 event script processing begins
Jun 22 18:06:39 reading event details
Jun 22 18:06:39 testing event details
Jun 22 18:06:39 running event script 'poesched.slax'
Jun 22 18:06:39 opening event script
'/var/db/scripts/event/poesched.slax'
Jun 22 18:06:39 reading event script 'poesched.slax'
Jun 22 18:06:51 event script output
```

```
Jun 22 18:06:51 begin dump
<?xml version="1.0"?>
Jun 22 18:06:51 end dump
Jun 22 18:06:51 inspecting event output 'poesched.slax'
Jun 22 18:06:51 finished event script 'poesched.slax'
Jun 22 18:06:51 event script processing ends
...

root> show interfaces terse
Interface               Admin Link Proto    Local
Remote
ge-0/0/0                up    up
ge-0/0/0.0              up    up   eth-switch
ge-0/0/1                up    up
ge-0/0/1.0              up    up   eth-switch
ge-0/0/2                up    up
ge-0/0/2.0              up    up   eth-switch
ge-0/0/3                up    up
ge-0/0/3.0              up    up   eth-switch
...
```

Listing 22. Verifying that interfaces are enabled correctly after disabling them

The script was tested with enabling and disabling PoE ports and everything worked without any problems. This script can be used in business environments to shut down PoE devices to reduce energy consumption. The script is easy to set up and maintain so there should not be any problems using it in larger scale installations. The script is also very simple; it just shuts down and enables ports at a certain time of the day. Nevertheless, environments can be different and there can be a need for customizations sometimes, and therefore customizing the script is next explained briefly.

Modifying Junos Scripts

Sometimes there can be special needs and it is not enough to just shut down ports at a certain time of the day. One example could be a company where workers usually stop working at a certain time but there can be an ongoing call preventing someone to leave the office. It would not be a desired functionality if a script turned off an IP-phone during a call. In a case like this the script could be modified to check if the interface is in use and the shutdown process could be cancelled for that port if there is an ongoing call.

It is easy to determine if there is an ongoing call. Depending on the protocol and settings that the phone is using to establish a phone call, different amounts of packets are transmitted through the network interface [22]. Junos XML API includes a command to retrieve interface information with PPS (Packets per Second) values. A SLAX script to demonstrate how to retrieve PPS values from an interface is shown in listing 23 below.

```
version 1.0;

ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";

import "../import/junos.xsl";

match / {
            <op-script-results> {
                        var $stat-rpc = <get-interface-
information>;
                        var $stat = jcs:invoke($stat-rpc);
                        var $con = jcs:open();

                        if (not($con)) {
                                    <xnm:error> {
                                                <message> "Not
able to connect with local mgd";
                                    }
                        }

                        var $test = $stat/physical-interface[name
== "ge-0/0/15"]/traffic-statistics/output-pps;
                        <output> $test;
            }
}
```

Listing 23. OP script to display ge-0/0/15 interface's PPS information

The above script displays the output PPS of the interface "ge-0/0/15" to the command line when it is executed. This PPS information can be used in the PoE scheduler script in an "if" statement to check if the PPS value exceeds a certain threshold. Other information can be used in the same way to modify the PoE scheduler script for different needs.

Energy Automation with Cisco Scripting

The tested Cisco device was a Cisco Catalyst network switch model WS-C3560V2-24PS-E. During the testing the switch was equipped with IOS version 12.2(55)SE7. The script which was tested was written to have similar functionalities as the PoE scheduler script provided by Juniper. The original Tcl script was developed in a virtual environment and when testing with a C3560 switch, some problems were encountered. First when trying to run the script using EEM, it did not work. This was because EEM runs scripts in a protected mode so EXEC mode commands are not directly accessible. This problem was solved by opening a CLI connection separately with functions from Cisco EEM namespace.

Another problem was that the version of IOS in the tested device did not recognize the “section” command so the whole configuration would have to be parsed by the script instead of one section. Rather than parsing the whole configuration a decision was made to use a different command to retrieve interface information. As a result, the final script does not include an option to create multiple schedule groups. This is yet just another example of the fragmentation of the Cisco IOS.

The developed Tcl script is executed by a user assignable timer which is recommended to be set to one hour. By default the script shuts down all of the ports but individual ports can be excluded. All settings for the script are saved in environment variables in the configuration. First the script retrieves all the environment variables and the system time and sets the interface type. This initialization phase is shown in listing 24.

```
::cisco::eem::event_register_timer watchdog name timer\
              time $EEM_POE_INTERVAL

# Import eem namespace
namespace import ::cisco::eem::*

# Set start and end hours from environment variables
set start_hour $EEM_POE_START_HOUR
set end_hour $EEM_POE_END_HOUR

# Set interface exclude from environment variables
# Interfaces have to be comma separated interface numbers
set interface_exclude $EEM_POE_INTERFACE_EXCLUDE
set interface_exclude [split $interface_exclude ,]

# Set interface type
set interface_type "FastEthernet"

# Get current time
set current_time [clock format [clock seconds] -format "%H"]
```

Listing 24. Initializing Tcl script

After initialization the script creates a CLI connection and sends a command to the device to get a list of the interfaces. This is done in three phases. First the CLI connection is opened and after that the command is sent. Finally the result of the command is read from the CLI. All these steps have to be done separately since there is no way to send EXEC commands straight to the device when the script is executed through EEM. Creating the CLI connection and retrieving interfaces is shown in listing 25.

```
# Open cli connection
if [catch {cli_open} result] {
error $result $errorInfo
} else {
array set cli1 $result
}

# Issue a command to cli to get interfaces
if [catch {cli_write $cli1(fd) "show ip interface brief | include
$interface_type"} result] {
error $result $errorInfo
}

# Read the answer from cli
if [catch {cli_read $cli1(fd) } result] {
error $result $errorInfo
} else {
set interfaces_temp $result
}
```

Listing 25. Creating a CLI connection and retrieving interface names

Interfaces are retrieved with the ”show ip interface brief” command which returns the interface names with a lot of other information. Interface names have to be parsed from the output. First whitespaces and other invisible symbols are removed with regexp, and then the list of interface names with other information is parsed inside a “foreach” statement. Interfaces are saved in a list if they are not excluded from the configuration. The code to parse the CLI output is shown in listing 26.

```
# Remove extra spaces and line breaks
set interfaces_temp [regexp -all -inline {\S+} $interfaces_temp]

# Parse interface list and save interfaces to a variable
set interfaces {}
set exclude 0
foreach x $interfaces_temp {
            if { [string range $x 0 [expr {[string length $inter-
face_type]-1}]] == $interface_type } {
                        # Exclude interfaces based on user set
variable
                        foreach y $interface_exclude {
                                    if { [string range $x [string
length $interface_type] [string length $x]] == $y} {
                                                set exclude 1
                                    }
                        }
                        if { $exclude == 0 } {
                                    lappend interfaces $x
                        } else {
                                    set exclude 0
                        }
            }
}
```

Listing 26. Parsing results from CLI

After parsing the interface names, the script has now a list of all the interfaces which have to be enabled or disabled. The last part of the script uses the previously opened CLI connection and enters the configuration mode. The current time is checked against the configuration and the script enables or disables ports if they are set to be disabled or enabled at that time in the configuration. Listing 27 shows how the script enters the configuration mode and checks if ports need to be enabled or disabled.

```
# Enter configuration mode
 if [catch {cli_exec $cli1(fd) "en"} result] {
     error $result $errorInfo
 }
 if [catch {cli_exec $cli1(fd) "config t"} result] {
     error $result $errorInfo
 }

# Disable or enable interfaces if schedule is met
foreach x $interfaces {
            if { $current_time == $start_hour } {
                        if [catch {cli_exec $cli1(fd) "interface
$x"} result] {
          error $result $errorInfo
                        }
                        if [catch {cli_exec $cli1(fd) "no shut-
down"} result] {
          error $result $errorInfo
                        }
                        if [catch {cli_exec $cli1(fd) "exit"} re-
sult] {
          error $result $errorInfo
                        }
                        puts "bringing up interface $x"
            }
...
}
```

Listing 27. Modifying configuration and changing interface status in the script

Testing the script

The script folder in EEM can be set by the user. Tcl scripts for EEM are called policies. During testing, the device user policy folder was set to be the flash drive's root folder. Cisco also supports SCP so it was used to transfer the script to the device. Also the FTP server could be used to execute a script from a remote location. Transferring the script from the Unix shell using SCP is shown in the listing 28.

```
user@ubuntu:~/ciscotcl$ scp poesched.tcl cis-
co@10.10.10.1:/poesched.tcl
Password:
poesched.tcl
100% 2369     2.3KB/s   00:00
```

Listing 28. Transferring the script to a Cisco device with Secure Copy (SCP)

After transferring the script, the script had to be enabled and a few variables had to be set for the script to work. The first four lines in listing 29 are environment variables to set up the script. The first line sets the interval in seconds after the script is executed. The next variable is to indicate the interfaces which are excluded from the script. Interface numbers have to be separated by a comma. The next two variables are used to set the starting hour and ending hour for disabling and enabling ports. After variables, the user policy folder is set to tell the system where to search for Tcl scripts. The last line enables the script that was transferred.

```
event manager environment EEM_POE_INTERVAL 3600
event manager environment EEM_POE_INTERFACE_EXCLUDE 0/4,0/24
event manager environment EEM_POE_START_HOUR 18
event manager environment EEM_POE_END_HOUR 6
event manager directory user policy "flash:/"
event manager policy poesched.tcl type user
```

Listing 29. EEM variables and settings

Interfaces from “fa0/1” to “fa0/4” were used for testing. IP-phones were connected to ports “fa0/1” and “fa0/2” and wireless access points to ports “fa0/3” and “fa0/4”. The interface list before the script execution can be seen in listing 30. All ports were enabled before script execution and port “fa0/4” was in the exclude list so it should not be disabled after the script execution.

```
SW1#show ip int bri
Interface              IP-Address      OK? Method Status
Protocol
Vlan1                  unassigned      YES NVRAM  up
up
FastEthernet0/1        unassigned      YES unset  up
up
FastEthernet0/2        unassigned      YES unset  up
up
FastEthernet0/3        unassigned      YES unset  up
up
FastEthernet0/4        unassigned      YES unset  up
up
FastEthernet0/5        unassigned      YES unset  down
down
...
```

Listing 30. Interface list before Tcl script execution

The time interval was modified to be one minute using the variable “EEM_POE_INTERVAL” to test the script execution. Listing 31 shows how the “EEM_POE_END_HOUR” variable was set to correspond the system time in order for the script to shut down ports. After the script was executed, debugging information was shown in the console indicating that the script was executed correctly.

```
SW1#show clock
16:36:02.054 UTC Wed Jul 16 2014

SW1#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
SW1(config)#event manager environment EEM_POE_END_HOUR 16
SW1(config)#end

SW1#
Jul 16 16:37:03.635: %HA_EM-6-LOG: poesched.tcl: shutting down in-
terface FastEthernet0/1
Jul 16 16:37:04.197: %HA_EM-6-LOG: poesched.tcl: shutting down in-
terface FastEthernet0/2
Jul 16 16:37:04.767: %HA_EM-6-LOG: poesched.tcl: shutting down in-
terface FastEthernet0/3
...
Jul 16 16:37:14.766: %SYS-5-CONFIG_I: Configured from console by
vty0
...
```

Listing 31. Executing Tcl script to shut down interfaces

The console debug information shows that ports from “fa0/1” to “fa0/3” were disabled. Interface “fa0/4” was in the exclude list so it remained up. To make sure that the interfaces were really disabled, the interface list had to be checked. Listing 32 shows the interface list after the script execution.

```
SW1#show ip int bri
Interface              IP-Address      OK? Method Status
Protocol
Vlan1                  unassigned      YES NVRAM  up
up
FastEthernet0/1        unassigned      YES unset  administratively
down down
FastEthernet0/2        unassigned      YES unset  administratively
down down
FastEthernet0/3        unassigned      YES unset  administratively
down down
FastEthernet0/4        unassigned      YES unset  up
up
FastEthernet0/5        unassigned      YES unset  administratively
down down
...
```

Listing 32. Interface list after first Tcl script execution

The first three ports were disabled and “fa0/4” was left open because it was in the exclude list. Everything worked appropriately and the next step was to verify that the script also brings the interfaces up. For that, the start hour variable needed to be modified. Listing 33 shows how the “EEM_POE_START_HOUR” variable was set to correspond to the system time. After changing the variable, the script was executed and log messages appeared in the console.

```
SW1#show clock
16:41:27.221 UTC Wed Jul 16 2014

SW1#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
SW1(config)#event manager environment EEM_POE_END_HOUR 18
SW1(config)#event manager environment EEM_POE_START_HOUR 16
SW1(config)#end

SW1#
Jul 16 16:42:03.636: %HA_EM-6-LOG: poesched.tcl: bringing up in-
terface FastEthernet0/1
Jul 16 16:42:04.198: %HA_EM-6-LOG: poesched.tcl: bringing up in-
terface FastEthernet0/2
Jul 16 16:42:04.760: %HA_EM-6-LOG: poesched.tcl: bringing up in-
terface FastEthernet0/3
...
Jul 16 16:42:15.699: %SYS-5-CONFIG_I: Configured from console by
vty0
...
SW1#
```

Listing 33. Executing Tcl script to enable interfaces

The execution log shows that the interfaces were brought up correctly. Only one check remained to make sure from the interface list that the interfaces were enabled. Listing 34 shows the interface list after the second execution of the script.

```
SW1#show ip int brief
Interface              IP-Address      OK? Method Status
Protocol
Vlan1                  unassigned      YES NVRAM  up
up
FastEthernet0/1        unassigned      YES unset  up
up
FastEthernet0/2        unassigned      YES unset  up
up
FastEthernet0/3        unassigned      YES unset  up
up
FastEthernet0/4        unassigned      YES unset  up
up
FastEthernet0/5        unassigned      YES unset  down
down
```

Listing 34. Interface list after second Tcl script execution

All steps were performed without errors. The script was tested, and after modifying the interval, start hour and end hour variables to the desired values, the script was ready to be used. The Tcl script differs a little from the SLAX script but offers almost the same functionality. Both scripts are easy to maintain and deploy. The next chapter will explain briefly how to modify the script.

Modifying Cisco Scripts

Modifying Cisco Tcl scripts and adding features requires far more work than modifying SLAX scripts. This is because with a Cisco script the configuration or system information can only be viewed like it is presented in the CLI. It means that all the output from the system has to be parsed with regular expressions to find the right information. SLAX is XML based so navigating through the XML hierarchy and finding the right information is fairly easy. XML Programmatic Interface (XML-PI) was released by Cisco and included in the IOS version 12.4(22)T. XML-PI transforms CLI output to XML format [23]. Unfortunately all of the Cisco devices are not supported. For example, the devices mentioned in the testing part of this paper cannot be updated to an IOS version which supports XML-PI.

Interface PPS information could be retrieved for example with a “show interfaces [interface name]” command. This command would show the average input and output rate from the last five minutes. There are several commands which can be used to parse the output. The command “regexp” can be used to parse the output using a regular expression. The command output can be saved to a list and then iterated through the list to find the right information. When iterating through the list, a command like “string range” can be used to match a string range from the list.

5.3 Power Measurements

In the thesis power measurements were made by powering two IP-phones and two wireless access points with a switch and then turning off the ports. The first measurements were made before shutting down the ports and the second measurements after shutting down the ports. All measurements were made twice to minimize measurement errors. Power consumption was observed during one minute and the fluctuation range was recorded from that time period.

IP-phones used for testing:

- Cisco IP Phone 7960
- Cisco IP Phone 7912

Wireless access points used for testing:

- Cisco AIR-AP1231G-E-K9
- Cisco AIR-AP1230A-A-K9

The power measurements were done with Fluke 43B Power Quality Analyzer. The power measurements might contain a small error margin because of changing current but relatively the measurements are reliable. The aim was to calculate how much energy can be saved, so absolute figures are not needed and a relative result is enough.

Table 2. Reported power consumption from the manufacturer [24; 25]

Device	**Power consumption**
Cisco IP Phone 7960	6.3W
Cisco IP Phone 7912	6.3W
Cisco AIR-AP1231G-E-K9	13.0W (maximum)
Cisco AIR-AP1230A-A-K9	13.0W (maximum)
Sum	**38.6W**

Table 2 shows the manufacturer's reported power consumption for the tested devices. The sum for all the devices is a little below 40 watts. Expected consumption in power measurements should be below this since there is no load on the devices during power measurements. The test devices are just powered up so they should consume a minimal amount of energy. Of course energy efficiency and power management features of the switch also affect the power consumption.

Table 3. Power measurement results

Device	**First measurement**	**Second measurement**
Cisco	35.0W – 37.3W	53.6W – 55.9W
Juniper	123.5W – 125.8W	172.4W – 174.8W

The results of the power measurements can be seen in table 3. Without any load the Juniper switch uses much more power than the Cisco switch. With the PoE devices connected, the Juniper switch seems to consume relatively more power than the Cisco switch. The Juniper switch is more powerful and designed to be used in larger scale installations than the Cisco switch. This can also be seen in the power consumption. Nevertheless, Cisco seems to have better power management features since the power consumption does not increase so much when PoE devices are connected.

Table 4. Power consumption difference

Device	**Power consumption difference**
Cisco	18.6W
Juniper	48.9W – 49.0W

Table 4 shows the difference between the first and the second power measurements. The results in table 4 can be interpreted as the saved energy when the devices are shut down. This test was carried out only with four PoE devices connected so real world energy savings can be much higher. The Juniper switch seems to have a very high overhead in the power consumption and the Cisco switch seems to use about the same amount of power that the devices are expected to use when they are idle. The Juniper switch on the other hand uses much more than the maximum power intake these devices have. This difference could be diminished if all the PoE ports were in use.

6 Energy Saving Calculations

This chapter will introduce energy saving calculations for an example office of 200 employees. Estimated energy consumption of all the PoE devices will be calculated with the estimated energy savings when energy automation is deployed in the company. Possible savings will also be calculated with the average electricity market price in Finland (in year 2014).

The network of the example office consists of 12 PoE switches with 48 ports and two core switches. The office consists of two floors each with 100 workstations. The workstation computers and IP-phones are connected to the access level PoE switches. The network also includes ten wireless access points with five access points in both floors. Figure 5 below shows a very simplified network diagram of the example office.

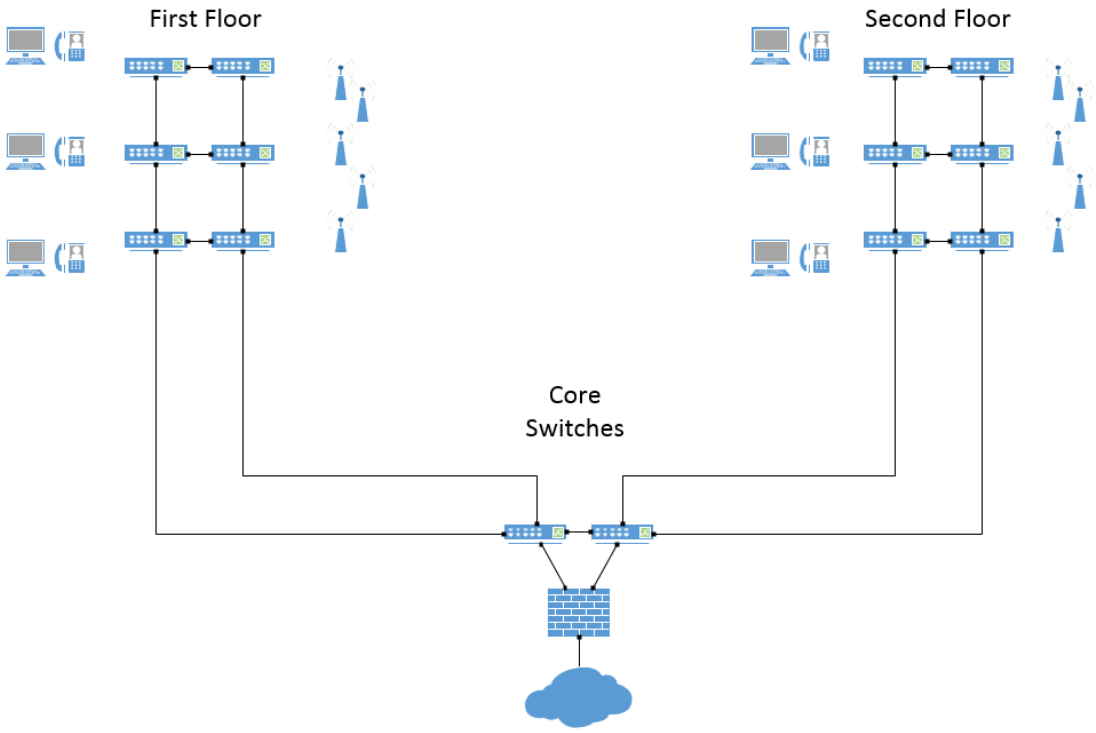

Figure 5. Example office network diagram

A network switch can consume different amounts of energy depending on its features and network load. For example as observed in the previous chapter, the Cisco switch's power consumption was less than 40 watts when the Juniper switch was using more than 100 watts when it was idle. A rough average power consumption estimate for a network switch is 60 watts. This would mean that in the example company, all the

switches together would consume 840 watts of energy every hour. Modern IP-phones consume roughly 2-4 watts and wireless access points about 5-10 watts per hour. In the example company it would mean that PoE devices would consume 450-900 watts per hour. Basically this means that 450 to 900 watts of energy is wasted during the hours when nobody is at the office. It could be said that in a typical office there is rarely anyone working after 8 pm or before 6 am. The estimate of saved energy is calculated in table 5 below with the assumption that PoE devices are shut down between 8 pm and 6 am. Estimated money savings were calculated based on the energy price (5.86 c/kWh) set by the Fortum energy company for a time period from 1st October to 31st December 2014 [26].

Table 5. Energy saving calculations

Time frame	**Saved energy**	**Saved money**
Day	4.5 – 9.0 kWh	€ 0.26 – 0.53
Three months	405.0 – 810.0 kWh	€ 23.70 – 47.50
Year	1642.5 – 3285.0 kWh	€ 96.3 – 192.5

With only shutting down wireless access points and IP-phones the energy savings can be thousands of kilowatts. If energy automation was taken further to the network switches and workstations this could be much more. Money savings also rise to hundreds of euros on a yearly basis. A company with several offices like this could save thousands of euros. Network switches in the example office use about the same amount of energy as the PoE devices attached to them. Energy savings could be doubled by also shutting down the network switches.

7 Discussion

This thesis has introduced very simple scripts to shut down PoE ports at a certain time of the day. These scripts could be modified to support additional features and tailor them for different needs. For example the time scheduling is not perfect in the scripts. Weekends could also be considered as many offices are empty during weekends. More automation could also be built apart from the scripts. A webpage could be created to manually execute scripts, change timeframes or monitor port statuses. Even a smart phone application could be created quite easily to connect to the XML API interface of the devices to do the same tasks. Nevertheless these simple scripts alone are an effective way to reduce energy consumption in a company.

In the project, both tested scripts worked as they were meant to and power measurements proved that it is possible to save energy just by shutting down PoE ports. The tested scripts are easy to deploy and maintain so they can also be used in a bigger scale installment. Anyone can easily utilize these scripts and start saving energy in their network environment. Deploying the scripts should not take longer than one day. The only problems encountered were related to Cisco IOS when all the commands were not supported. Nevertheless after some changes to the script, a solution was found. Both platforms offer a very extensive scripting support for different needs. SLAX might need some getting used to but it is very logical and seems to be better integrated to the system than Tcl. Tcl on the other hand is easier to use for someone with previous programming experience. Testing and power measurements were done with just one network device and only four PoE devices were attached to them. In larger scale installments the energy savings will become noticeably bigger.

Devices which can be automated do not necessarily have to be the devices introduced in this paper. Network devices from other manufacturers could also be used. In fact any device connected to the network could be included in the energy automation, devices like computers and printers for instance. Even the network switches could be shut down and then turned on again. In this case only shutting down PoE ports would not be enough, but there already exist technologies that could be used such as Wake-on-LAN or smart power outlets. Wake-on-LAN is a networking standard that allows computing devices to be turned on by a network message. There are several smart power outlets which can be connected to the Ethernet network and controlled by a network message. One of the simplest ones cuts the power transmission when it receives a ping packet.

When using smart power outlets, devices like monitors which are not directly connected to the network could also be turned off automatically. This could enable even bigger energy savings.

This thesis has only covered energy automation for PoE devices, but including all the network equipment, office peripheral devices and computers in energy automation could result in much higher energy savings. There are many existing technologies that can be used and new energy saving technologies are introduced all the time. Focus should not only be on the power consumption of the devices, but also it is necessary to consider when the devices are actually needed and when they need to be powered on.

8 Summary

The solution introduced in this paper proved to be very easy and fast to deploy. It takes less than half an hour to enable the ready-made script in one network device. Once the script was tested and issues within the script were identified, there were no other technical challenges. As the calculations showed, money could also be saved when enabling energy automation for PoE devices. These figures might not be that big in corporate world, and only financially the change could be hard to justify. Nevertheless the energy savings are still big and benefits for a company do not need to be only calculated in hard cash. A company could for example advertise itself with green offices where also network devices are considered when saving energy and preserving the environment.

Overall the goals that were set in the beginning were reached. Simple and easily deployable scripts were made to shut down and power up PoE devices in order to reduce energy consumption. Scripts were proven to function as they should and power measurements proved the rather obvious fact that energy consumption can be effectively decreased. Calculations made for an example office also show that energy savings can be quite big. This work also leaves an open path for further improvements and research subjects. Energy automation could be extended to other devices and the solution proposed in this paper could be improved to support additional features. The main goal was to have a simple solution that can be easily deployed to network environments and companies. That goal was reached and any company could benefit from the energy automation solution introduced in this thesis.

References

1 Bolla R., Bruschi R., Davoli F., Cucchietti F. Energy Efficiency in the Future Internet: A Survey of Existing Approaches and Trends in Energy-Aware Fixed Network Infrastructures. University of Genoa, Research Unit of Genoa, Telecom Italia; 2009.

2 Juniper Networks. Company History [online]. Juniper Networks; URL: https://www.juniper.net/us/en/company/profile/history/. Accessed 21 February 2014.

3 Juniper Networks. Junos XML API and Junos XML Management Protocol Overview [online]. Juniper Networks; 27 June 2012. URL: https://www.juniper.net/techpubs/en_US/junos12.2/topics/concept/junos-script-automation-junos-xml-protocol-and-api-overview.html. Accessed 21 February 2014.

4 Duffy J. Juniper acquires NetScreen [online]. Network World; 9 February 2004. URL: https://www.networkworld.com/edge/news/2004/0209juniscreen.html. Accessed 21 February 2014.

5 Cameron R. Junos Security. California: O'Reilly Media, Inc.; 2010.

6 Juniper Networks. Security Features for EX Series Switches Overview [online]. Juniper Networks; 14 November 2010. URL: https://www.juniper.net/techpubs/en_US/junos10.4/topics/concept/ex-series-security-overview.html. Accessed 21 February 2014.

7 Call C. This Week: Applying Junos Automation. USA: Vervante Corporation; 2011.

8 Juniper Networks. Junos OS CLI User Guide. Sunnyvale, California: Juniper Networks; 2012.

9 Kurose J.F, Ross K.W. Computer Networking. USA: Pearson Education, Inc.; 2010.

10 Brocade Communications Systems. Company Highlights [online]. Brocade Communications Systems. URL: http://www.brocade.com/downloads/documents/company_information/company-highlights-fl.pdf. Accessed 10 May 2014.

11 Machowinski M. Enterprise Networking and Communication Vendor Leadership Scorecard. USA: Infonetics Research, Inc.; 2013.

12 Cisco Systems, Inc. Cisco IOS XR XML API Guide. San Jose, USA: Cisco Systems, Inc; 2011.

13 Cisco Systems, Inc. Cisco IOS IP Configuration Guide. San Jose, USA: Cisco Systems, Inc; 2006.

14 Cisco Systems, Inc. Cisco Configuration Professional User Guide. San Jose, USA: Cisco Systems, Inc; 2011.

15 Cisco Systems, Inc. Enhanced Interior Gateway Routing Protocol (EIGRP) Informational RFC [online]. Cisco Systems, Inc; March 2013.
URL: http://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/enhanced-interior-gateway-routing-protocol-eigrp/qa_C67-726299.html. Accessed 9 April 2014.

16 World Wide Web Consortium. Extensible Markup Language (XML) 1.0 (Fifth Edition) [online]. World Wide Web Consortium; 7 February 2013.
URL: http://www.w3.org/TR/2008/REC-xml-20081126/. Accessed 18 April 2014.

17 Juniper Networks. SLAX Overview [online]. Juniper Networks; 5 November 2012.
URL: http://www.juniper.net/techpubs/en_US/junos12.3/topics/concept/junos-script-automation-slax-overview.html. Accessed 19 April 2014.

18 Shafer P. Libslax wiki [online]. GitHub; 21 August 2013.
URL: https://github.com/Juniper/libslax/wiki. Accessed 8 June 2014.

19 Blair R., Durai A., Lauttmann J. Tcl Scripting for Cisco IOS. Indianapolis, USA: Cisco Press; 2010.

20 Ylönen T. Lonvick C. RFC 4251: SSH Protocol Architecture [online]. IETF; January 2006. URL: https://tools.ietf.org/html/rfc4251. Accessed 10 May 2014.

21 Juniper Networks. poesched [online]. Juniper Networks; 2009.
URL: http://www.juniper.net/us/en/community/junos/script-automation/library/configuration/poesched/. Accessed 20 July 2014.

22 Cisco Systems, Inc. Voice Over IP - Per Call Bandwidth Consumption [online]. Cisco Systems, Inc.; 2 February 2006.
URL: http://www.juniper.net/us/en/community/junos/script-automation/library/configuration/poesched/. Accessed 2 August 2014.

23 Cisco Systems, Inc. EEM CLI Library XML-PI Tcl Support [online]. Cisco Systems, Inc.; 16 November 2011.
URL: http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/eem/configuration/12-4t/eem-12-4t-book/eem-cli-xml-pi-tcl.html/. Accessed 24 August 2014.

24 Cisco Systems, Inc. Power over Ethernet (PoE) Power Requirements FAQ [online]. Cisco Systems, Inc.; 26 June 2008.
URL: http://www.cisco.com/c/en/us/support/docs/voice-unified-communications/unified-ip-phone-7900-series/97869-poe-requirement-faq.html. Accessed 24 August 2014.

25 Cisco Systems, Inc. Cisco Aironet 1200 Series Access Point Data Sheet [online]. Cisco Systems, Inc.
URL: http://www.cisco.com/c/en/us/products/collateral/wireless/aironet-1200-access-point/product_data_sheet09186a00800937a6.html. Accessed 24 August 2014.

26 Fortum Oyj. Fortum Kesto, sähkön hinta [online]. Fortum Oyj; 9 September 2014.
URL: https://www.fortum.com/countries/fi/yksityisasiakkaat/hinnastot/sahkosopimuksen-hinta-kesto/pages/default.aspx. Accessed 13 September 2014.

Junos SLAX script

```
/*
 * Author        : Patricio Giecco
 * Version       : 1.0
 * Last Modified :
 * JUNOS Release : 9.3 and above
 * Platform      : EX Series
 *
 * Description   : poesched.slax
 * This event script is executed on the occurrence of 'poe-scheduler-start'
 * event. According to the configuration mentioned in this file, under
 * 'event-definition', this event is triggered in every 1 hour and this
 * script get executed. This script disables/enables some interfaces for
 * poe, based on the configured schedule for a particular time.
 */

version 1.0;

ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";

import "../import/junos.xsl";

var $event-definition = {
    <event-options> {
        <generate-event> {
            <name> 'poe-scheduler-start';
            <time-interval> '3600';
        }
        <policy> {
            <name> 'poe-scheduler';
            <events> 'poe-scheduler-start';
            <then> {
                <event-script> {
                    <name> 'poesched.slax';
                }
            }
        }
    }
}

var $arguments = <argument> {
    <name> "time";
    <description> "execute the scheduled action at the specified time";
}

/*
 * This script is executed on the occurrence of event 'poe-scheduler-start'
 * which occurs in every one hour. On execution of this script POE on/off
 * on some ports based on a configured schedule. If the time was passed as
 * a parameter then use the passed value otherwise get the system time-date
 * and extract the time portion of it i.e. current time (only the hour).
 */
param $time = {
    var $uptime = jcs:invoke("get-system-uptime-information");
    var $date = $uptime//current-time/date-time;

    expr substring($date, 12, 2);
}
```

```
match / {

    /*
     * Get the poe configuration
     */
    var $poe-rpc = <get-configuration> {
        <configuration> {
            <poe>;
        }
    }

    var $poe = jcs:invoke($poe-rpc);

    /*
     * Open a connection with mgd
     */
    var $con = jcs:open();

    if (not($con)) {
        <xnm:error> {
            <message> "Not able to connect with local mgd";
        }
    }

    /*
     * Generate the xml formated config delta
     */
    var $xml := {
        <configuration> {
            <poe> {
                /*
                 * Go through all the groups that are scheduled at this time
                 */
                for-each ($poe//apply-macro[starts-with(name, "scheduler-") &&
                          data/value == number($time)]) {
                    var $splitName = jcs:regex("scheduler-(.+)", name);
                    var $group = $splitName[2];
                    var $action = data[value == number($time)]/name;

                    /*
                     * select all the interfaces belonging to the sceduled
                     * groups
                     */
                    for-each ($poe//interface[apply-macro/name == "scheduler"
                              && apply-macro/data/name == "group" &&
                              apply-macro/data/value == $group]) {
                        if ($action == "on" && disable) {
                            <interface> {
                                <name> name;
                                <disable delete = "delete">;
                            }
                        } else if ($action == "off" && not(disable)) {
                            <interface> {
                                <name> name;
                                <disable>;
                            }
                        }
                    }
                }
            }
        }
    }


    /*
```

```
     * If some interface statements are present in the $xml/configuration/poe,
     * then commit the changes otherwise not required
     */
    if ($xml/configuration/poe/interface) {
        var $results = {
            call jcs:load-configuration($connection = $con,
                                        $configuration = $xml);
        }
    }
}
```

Cisco Tcl script

```
#
# Author        : Olli Paakkunainen
# Version       : 1.0
# Last Modified : 13.7.2014
# IOS Release           : 12.2 and above
#
# Description   : poesched.tcl
# This script is executed every X seconds according to
# an user set environment variable. Script enables
# or disables interfaces based on an user set schelude.
# By default all FastEthernet ports are disabled but
# individual ports can be excluded from the script.
#

# Set time interval from environment variable
::cisco::eem::event_register_timer watchdog name timer\
              time $EEM_POE_INTERVAL

# Import eem namespace
namespace import ::cisco::eem::*

# Set start and end hours from environment variables
set start_hour $EEM_POE_START_HOUR
set end_hour $EEM_POE_END_HOUR

# Set interface exclude from environment variables
# Interfaces have to be comma separated interface numbers
set interface_exclude $EEM_POE_INTERFACE_EXCLUDE
set interface_exclude [split $interface_exclude ,]

# Set interface type
set interface_type "FastEthernet"

# Get current time
set current_time [clock format [clock seconds] -format "%H"]

# Open cli connection
if [catch {cli_open} result] {
error $result $errorInfo
} else {
array set cli1 $result
}

# Issue a command to cli to get interfaces
if [catch {cli_write $cli1(fd) "show ip interface brief | include $inter-
face_type"} result] {
error $result $errorInfo
}

# Read the answer from cli
if [catch {cli_read $cli1(fd) } result] {
error $result $errorInfo
} else {
set interfaces_temp $result
}

# Remove extra spaces and line breaks
set interfaces_temp [regexp -all -inline {\S+} $interfaces_temp]

# Parse interface list and save interfaces to a variable
set interfaces {}
set exclude 0
```

```
foreach x $interfaces_temp {
          if { [string range $x 0 [expr {[string length $interface_type]-1}]]
== $interface_type } {
                        # Exclude interfaces based on user set variable
                        foreach y $interface_exclude {
                                    if { [string range $x [string length $in-
terface_type] [string length $x]] == $y} {
                                                set exclude 1
                                    }
                        }
                        if { $exclude == 0 } {
                                    lappend interfaces $x
                        } else {
                                    set exclude 0
                        }
            }
}

# Enter configuration mode
 if [catch {cli_exec $cli1(fd) "en"} result] {
     error $result $errorInfo
 }
 if [catch {cli_exec $cli1(fd) "config t"} result] {
     error $result $errorInfo
 }

# Disable or enable interfaces if schedule is met
foreach x $interfaces {
            if { $current_time == $start_hour } {
                        if [catch {cli_exec $cli1(fd) "interface $x"} result]
{
         error $result $errorInfo
                        }
                        if [catch {cli_exec $cli1(fd) "no shutdown"} result] {
         error $result $errorInfo
                        }
                        if [catch {cli_exec $cli1(fd) "exit"} result] {
         error $result $errorInfo
                        }
                        puts "bringing up interface $x"
            }
            if { $current_time == $end_hour } {
                        if [catch {cli_exec $cli1(fd) "interface $x"} result]
{
         error $result $errorInfo
                        }
                        if [catch {cli_exec $cli1(fd) "shutdown"} result] {
         error $result $errorInfo
                        }
                        if [catch {cli_exec $cli1(fd) "exit"} result] {
         error $result $errorInfo
                        }
                        puts "shutting down interface $x"
            }
}
```